Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Matemática

Tarea Parte 1

Problema de mínimos cuadrados no lineal

Integrantes: Gonzalo Claro
Profesor: Alejandro Jofré
Auxiliares: Felipe Atenas
Daniel Pereda
Ayudantes: Makarena Fuentealba
Carlos Poblete
Vicente Rojas

Fecha de realización: 25 de junio de 2019
Fecha de entrega: 25 de junio de 2019
Santiago, Chile

Índice de Contenidos

1. Introducción **ii**

2. Parte a) **iii**

3. Parte b) **ix**

4. Parte c) **xiii**

Índice de Figuras

1. Ajuste mediante el método Newton-Raphson para los datos, en azul ydata inicial y en rojo el ajuste. viii
2. Ajuste mediante *leastsq* para los datos, en azul ydata inicial y en rojo el ajuste. . . viii
3. Ajuste mediante el método de Gauss-Newton para los datos, en azul ydata inicial y en rojo el ajuste. xii

Índice de Tablas

1. Tabla de los errores porcentuales para x^* a distintos valores de x_0 utilizando Newton-Raphson v
2. Tabla de los errores porcentuales de x^* para un ajuste al mismo valor de x_0 pero distinta cantidad de iteraciones vi
3. Tabla de los errores porcentuales de x^* para distintos valores de x_0 utilizando leastsq vii
4. Tabla de los errores porcentuales de x^* para distintos valores de x_0 utilizando el método de Gauss-Newton xi
5. Tabla de los errores porcentuales de x^* para un ajuste al mismo valor de x_0 pero distinta cantidad de iteraciones xii

1. Introducción

Esta tarea de optimización parte 1, tiene por objetivo introducir el algoritmo de Newton para la resolución del problema de mínimos cuadrados no lineal.

$$min_{x\in\mathbb{R}^n} \sum_{i=1}^{m} (r_{i(x)})^2$$

Primero en la parte a.1) se trata de buscar la solución a este problema mediante el método de Newton, el cual busca una aproximación para $r(x)$ en torno al punto x_k. Se busca la solución a $min\, \|r(x)\|^2$ or lo que iterando este método se ajustan los datos generados.

$$x_{k+1} = x_k - (J(x_k)^T J(x_k) + S(x_k))^{-1} J(x_k)^T r(x_k) \tag{1}$$

Donde:

$$r_i(x) = y_i - M(x, t_i) \tag{2}$$

$$M(x,t) = a \sin(\omega t + \phi) + bt \tag{3}$$

$$x = (a, \omega, \phi, b) \in \mathbb{R}^4$$

$y_{i=1}^m$ son mediciones asociadas a los tiempos $t_{i=1}^m$

Luego para la parte a.2) se intenta resolver el mismo problema pero mediante la función predeterminada *leastsq* del paquete optimize la biblioteca Scipy. La cual recibe como parámetros a la función $r(x)$ (que representa la diferencia entre los datos obtenidos y el modelo M) y un vector de valores iniciales x_0 para que comience desde ese punto el ajuste. Posteriormente se grafican tanto los resultados de la parte a.1) del método de Newton como de a.2) de la función predeterminada.

En la parte b) de la tarea se asume que la matriz $J(x)$ es invertible para todo x se puede utilizar la dirección de descenso

$$-(J(x_k)^T J(x_k))^{-1} J(x_k)^T r(x_k)$$

En donde en vez de considerar la función r se usa su aproximación de Taylor de primer orden L^k en torno al punto x_k. En otras palabras se resuelve el problema $min \left\| L^k(x) \right\|^2$ por lo que la iteración para ajustar los datos pasa a ser

$$x_{k+1} = x_k - (J(x_k)^T J(x_k))^{-1} J(x_k)^T r(x_k) \tag{4}$$

Para ello se implementa una solución similar a la parte a.1), con este nuevo método alternativo y utilizando el mismo modelo de la parte a)

2. Parte a)

Para resolver la parte **a.1)** primero nos definimos nuevas variables iniciales, las cuales para el funcionamiento del método deben ser cercanas a los valores objetivos, por lo que las definiremos como:

$$x_0 = [a_0, \omega_0, \phi_0, b_0] = [a * 1.001, \omega * 1.001, \phi * 1.001, b * 1.001]$$

$$x_0 = [8.98374641, 1.01897537, 0.5370046, 0.32088405]$$

Recordemos que nuestro $x_{objetivo}$ segun nuestro rut (19390187) y nuestro número de lista (22), con 8 decimales es:

$$x = [a, \omega, \phi, b] = [8.97477164, 1.01795741, 0.53646814, 0.32056349]$$

Como vemos, la diferencia entre ambos vectores es pequeña:

$$x_0 - x = [0.00897477, 0.00101796, 0.00053647, 0.00032056]$$

Esto traducido en términos de error porcentual para cada componente, entre el valor real x y el valor utilizado x_0 es del orden de $0.0999\,\%$ para a, ω, ϕ y b.

Una vez tenemos nuestro vector de condiciones iniciales, procedemos a definir todas las funciones necesarias para implementar el metodo de la ecuación (1) y ver si podemos reducir este error de $0.0999\,\%$ para cada componente, encontrando tras el ajuste un x^* que se acerque al valor óptimo $x_{objetivo}$ con mayor precisión.

Primero definimos un nuevo modelo $M_o(x, t)$, el cual es idéntico al de la ecuación (3), pero lo definimos nuevamente ya que el modelo del archivo gendata tiene no tiene una dependencia para el vector x, sino que tiene directamente utilizados los datos del $x_{objetivo}$. Una vez se tiene definido el modelo $M_o(x, t)$, es directo definir $r(x)$ como $r_i(x) = y_i - M_o(x, t_i)$.

Ahora sólo nos faltan para definir la ecuación (1) el Jacobiano de $r(x)$ y la matriz S, matrices que tendrán dimensiones de 750x4 y 4x4 respectivamente. Para ello, las calculamos a mano.

El Jacobiano de $r(x)$ es:

$$J(r(x)) = \begin{bmatrix} -\sin(\omega t[0] + \phi) & -a\cos(\omega t[0] + \phi)t[0] & -a\cos(\omega t[0] + \phi) & -t[0] \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ -\sin(\omega t[750] + \phi) & -a\cos(\omega t[750] + \phi)t[750] & -a\cos(\omega t[750] + \phi) & -t[750] \end{bmatrix}$$

Y la matriz S es:

$$S(x) = \begin{bmatrix} 0 & \sum_{i=0}^{750} -r_i \cos(\omega t_i + \phi) t_i & \sum_{i=0}^{750} -r_i \cos(\omega t_i + \phi) & 0 \\ \sum_{i=0}^{750} -r_i \cos(\omega t_i + \phi) t_i & \sum_{i=0}^{750} r_i a t_i^2 \sin(\omega t_i + \phi) & \sum_{i=0}^{750} r_i a t_i \sin(\omega t_i + \phi) & 0 \\ \sum_{i=0}^{750} -r_i \cos(\omega t_i + \phi) & \sum_{i=0}^{750} r_i a t_i \sin(\omega t_i + \phi) & \sum_{i=0}^{750} r_i a \sin(\omega t_i + \phi) & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Para implementar tanto el tamaño de la matriz $J(x)$ como las sumatorias dentro de $S(x)$ se utilizaron ciclos for.

Una vez tenemos todo lo necesario para implementar la ecuación (1), procedemos a hacer tanto una función recursiva del método, como simples iteraciones del mismo. Para implementar la función recursiva, llamada $newton(x_0, x_{objetivo})$, que recibe un vector x_0 de condiciones iniciales y un vector $x_{objetivo}$ procedemos como sigue:

1) Calculamos $r(x_0)$, $J(x_0)$ y $S(x_0)$
2) Con estos valores realizamos la operación: $x_{k+1} = x_0 - (J(x_0)^T J(x_0) + S(x_0))^{-1} J(x_0)^T r(x_0)$
3) Si la diferencia (componente a componente) entre el vector iterado x_{k+1} y el vector objetivo x es pequeña, se retorna el valor de x_{k+1} que vendría a ser el x^* óptimo del ajuste e información acerca del número de iteraciones. En este paso puede imponer cualquier condición de parada que se estime conveniente, como que que el número de iteraciones sobrepase cierto límite (en este caso pondremos un máximo de 100 iteraciones), la condición de optimalidad de primer orden (ya que el problema es convexo e irrestricto) o que $\|\nabla f(x^*)\|$ sea muy pequeña.
4) Si no se cumple la condición impuesta en 3) el programa se ejecuta nuevamente desde 1) pero haciendo ahora $x_0 = x_{k+1} = x^*$ para que siga iterandose recursivamente hasta que cumpla con la condición del paso 3) y se detenga.

Si aplicamos esta iteración de $newton(x_0, x_{objetivo})$ para distintos valores de x_0 tenemos que:

Para $x_0 = x_{objetivo}$ el valor del ajuste tras 1 iteración es:
$x^* = [8.95525969, 1.01931438, 0.51372034, 0.32025165]$

Para $x_0 = 1.001 \cdot x_{objetivo}$ el valor del ajuste tras 1 iteración es:
$x^* = [8.95321482, 1.0193212, 0.51366947, 0.3202458]$

Para $x_0 = 1.1 \cdot x_{objetivo}$ el valor del ajuste tras 4 iteraciones es:
$x^* = [8.95843441, 1.01883108, 0.5205997, 0.32046822]$

Para $x_0 = 7 \cdot x_{objetivo}$ el método sobrepasa el límite de 100 iteraciones

Como para $x_0 = 7 \cdot x_{objetivo}$ sobrepasa el límite de iteraciones buscando el óptimo (no ha logrado cumplir con que la diferencia entre componentes sea pequeña), iteraremos 4 veces mediante un ciclo while, para ver el valor encontrado hasta ese punto por el método.

Para $x_0 = 7 \cdot x_{objetivo}$ el valor del ajuste tras 4 iteraciones es:
$x^* = [-0.13356736, 7.12176784, 3.95812959, 0.29381702]$

Para $x_0 = 7 \cdot x_{objetivo}$ el valor del ajuste tras 50 iteraciones es:
$x^* = [-0.07578891, 7.03376258, 5.67904635, 0.29385915]$

Si estos valores los comparamos con el valor original de x:

$$x = [a, \omega, \phi, b] = [8.97477164, 1.01795741, 0.53646814, 0.32056349]$$

y vemos el error porcentual de las aproximaciones del método de Newton para cada x_0, obtenemos la siguiente tabla:

Tabla 1: Tabla de los errores porcentuales para x^* a distintos valores de x_0 utilizando Newton-Raphson

Valor de x_0 utilizado	Error % para a	Error % para ω	Error % para ϕ	Error % para b	Iteraciones
$x_0 = x_{objetivo}$	0.2174	0.1333	4.2402	0.0972	1
$x_0 = 1.001 \cdot x_{objetivo}$	0.2401	0.1339	4.2497	0.0991	1
$x_0 = 1.1 \cdot x_{objetivo}$	0.1820	0.0858	2.9579	0.0297	4
$x_0 = 7 \cdot x_{objetivo}$	101.4882	599.6135	637.8126	8.3435	4

Un resultado interesante es que con el vector de valores iniciales $x_0 = 1.1 \cdot x_{objetivo}$, se obtiene la menor diferencia, y se podría deducir que para encontrar el mejor x_0 para el método, no sólo basta estar muy cerca de $x_{objetivo}$ para que funcione, sino lo suficientemente alejado también, ya que para los valores muy cercanos ($x_0 = x_{objetivo}$ y $x_0 = 1.001 \cdot x_{objetivo}$) el método tiene un error mayor. Pero esto probablemente se podría deber al número de iteraciones realizadas, ya que para el caso de $x_0 = 1.1 \cdot x_{objetivo}$ es de 4, mientras que para los otros 2 casos es de 1.

El número de iteraciones realizadas responde sólamente a las condiciones de parada, pero estas fueron definidas con no mucha exactitud, (diferencia de al menos 0.1 en la resta de los componentes) ya que si se ponía un valor de parada en que la diferencia entre x^* y $x_{objetivo}$ fuera menor a 0.1, el método se quedaba iterando infinitamente.

Para salir de esta interrogante, de si a mayor cantidad de iteraciones mejora el método, para el vector de condiciones iniciales más cercano al valor objetivo ($x_0 = 1.001 \cdot x_{objetivo}$) se realizaron "manualmente" más iteraciones mediante un ciclo while, para así ver que tanto mejoraba el resultado. Todo esto con el objetivo de ver si $x_0 = 1.001 \cdot x_{objetivo}$, que está más cerca del punto $x_{objetivo}$, "supera" a $x_0 = 1.1 \cdot x_{objetivo}$ que había obtenido los errores más bajos, pero con mayor cantidad de iteraciones, los resultados obtenidos fueron los siguientes:

Para $x_0 = 1.001 \cdot x_{objetivo}$ el valor del ajuste tras 1 iteración es:
$x^* = [8.95321482, 1.0193212, 0.51366947, 0.3202458]$

Para $x_0 = 1.001 \cdot x_{objetivo}$ el valor del ajuste tras 5 iteraciones es:
$x^* = [8.95537285, 1.01933231, 0.51344847, 0.32024498]$

Para $x_0 = 1.001 \cdot x_{objetivo}$ el valor del ajuste tras 15 iteraciones es:
$x^* = [8.95537285, 1.01933231, 0.51344847, 0.32024498]$

Vemos que el óptimo obtenido con el ajuste, x^*, se mantiene constante a partir de las 5 iteraciones aproximadamente. Esto se puede deber a que el largo del paso utilizado (el α_k) es pequeño. Esto puede implicar que se está cumpliendo alguna condición como la de Wolfe o Goldstein, y que el óptimo obtenido x^* es realmente un óptimo aproximado . si intentamos ver como evolucionó el error con más iteraciones tenemos la siguiente tabla:

Tabla 2: Tabla de los errores porcentuales de x^* para un ajuste al mismo valor de x_0 pero distinta cantidad de iteraciones

Valor de x_0 utilizado	Error % para a	Error % para ω	Error % para ϕ	Error % para b	Iteraciones
$x_0 = 1.001 \cdot x_{objetivo}$	0.2401	0.1339	4.2497	0.0991	1
$x_0 = 1.001 \cdot x_{objetivo}$	0.2161	0.1350	4.2909	0.0993	5
$x_0 = 1.001 \cdot x_{objetivo}$	0.2161	0.1350	4.2909	0.0993	15

Vemos que si bien el error disminuyó para la componente a, aumentó para el resto de componentes ω, ϕ y b. Esto nos permite concluir momentáneamente y en base a los resultados, que no necesariamente una mayor cantidad cantidad de iteraciones del método de Newton-Raphson lo hacen más preciso. También se puede concluir en base a esto, que no necesariamente entre más cerca se escoja x_0 de $x_{objetivo}$ será mejor el óptimo obtenido con el ajuste, x^*.

Para la parte **a.2)** se busca encontrar el ajuste x^* mediante la función predefinida *leastsq* del paquete optimize la biblioteca Scipy. Esta función como ya vimos, recibirá como parametros a la función $r(x)$ (que representa la diferencia entre los datos obtenidos $ydata_i$, y el modelo $M(x, t_i)$) y un vector de valores iniciales x_0 para que comenzar desde ese punto, por lo que el definirla es más o menos de la forma *leastsq(r,*x_0*)*.

Esta función *leastsq(r,x_0)* nos entregará distintos valores según el x_0 que se le entregue, por lo que veremos para los mismos valores de x_0 que le entregamos al método de Newton anteriormente cuáles son los ajustes para encontrar el óptimo x^*:

El valor de x^* para $x_0 = x_{objetivo}$ utilizando leastsq es:
$x^* = [8.95537285, 1.01933231, 0.51344851, 0.32024499]$

El valor de x^* para $x_0 = 1.001 \cdot x_{objetivo}$ utilizando leastsq es:
$x^* = [8.95537285, 1.01933231, 0.51344851, 0.32024499]$

El valor de x^* para $x_0 = 1.1 \cdot x_{objetivo}$ utilizando leastsq es:
$x^* = [8.95537285, 1.01933231, 0.51344847, 0.32024498]$

El valor de x^* para $x_0 = 7 \cdot x_{objetivo}$ utilizando leastsq es:
$x^* = [-0.14327059, 7.15029893, 3.351812, 0.29381029]$

Si llevamos estos resultados a una comparación con el valor óptimo u objetivo, $x_{objetivo}$ señalado anteriormente, tenemos esta tabla para los errores porcentuales de cada componente:

Tabla 3: Tabla de los errores porcentuales de x^* para distintos valores de x_0 utilizando leastsq

Valor de x_0 utilizado	Error % para a	Error % para ω	Error % para ϕ	Error % para b
$x_0 = x_{objetivo}$	0.2161	0.1350	4.2909	0.0993
$x_0 = 1.001 \cdot x_{objetivo}$	0.2161	0.1350	4.2909	0.0993
$x_0 = 1.1 \cdot x_{objetivo}$	0.2161	0.1350	4.2909	0.0993
$x_0 = 7 \cdot x_{objetivo}$	101.5963	602.4163	524.7923	8.3456

De acá se puede extraer otra conclusión muy interesante que no se comentó anteriormente y es que para valores muy lejanos a $x_{objetivo}$, como lo es $x_0 = 7 \cdot x_{objetivo}$, el método no funciona y entrega valores muy lejanos al óptimo. Por otra parte notamos que para valores relativamente cercanos al punto objetivo la función se comporta bien, tiene errores muy parecidos para cada uno por lo que nos da un margen mayor para encontrar x_0, a cambio de un error ligeramente mayor en las componentes que impide encontrar con mayor precisión el óptimo.

Si graficamos el resultado obtenido para el ajuste, en donde mediante $newton(x_0, x_{objetivo})$ se obtuvo el vector de condiciones iniciales óptimo x^* para $x_0 = 1.001 \cdot x_{objetivo}$. Usando $M_o(x^*, tdata)$ se obtuvo lo siguiente:

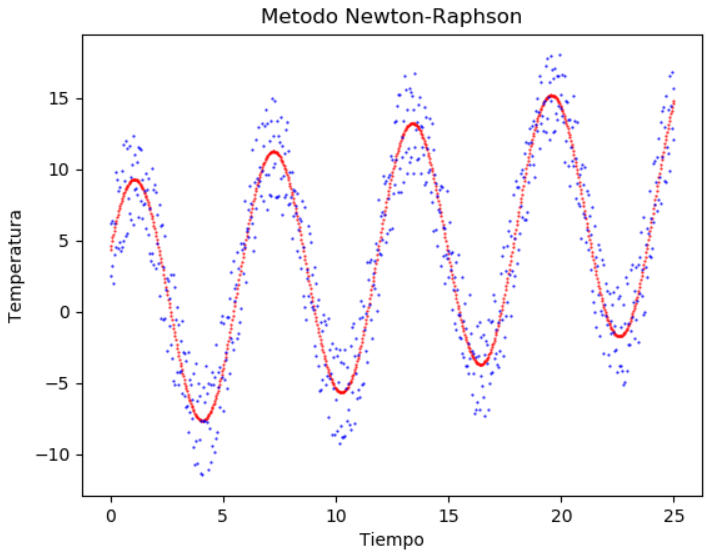

Figura 1: Ajuste mediante el método Newton-Raphson para los datos, en azul ydata inicial y en rojo el ajuste.

Por otra parte, para el ajuste mediante leastsq para el mismo valor de x_0 tenemos:

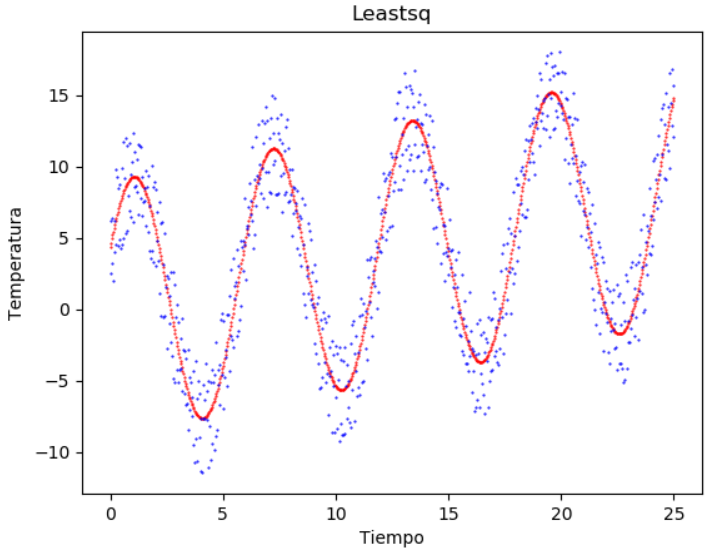

Figura 2: Ajuste mediante *leastsq* para los datos, en azul ydata inicial y en rojo el ajuste.

Vemos que tanto usando $newton(x_0, x_{objetivo})$ como con *leastsq* los datos se ajustan bastante bien y se parecen mucho, esto se debe a que ambos tienen un bajo nivel de error y parecido entre si para este valor de $x_0 = 1.001 \cdot x_{objetivo}$ (ver tablas 1 y 3).

3. Parte b)

Para la parte b) se resuelve el problema $min \left\| L^k(x) \right\|^2$ por lo que la iteración para ajustar los datos pasa a ser la ecuación (4)

$$x_{k+1} = x_k - (J(x_k)^T J(x_k))^{-1} J(x_k)^T r(x_k)$$

Para ello se implementó una solución muy similar a la de la parte a.1) y se vuelven a definir todas la funciones necesarias para implementar la iteración. Primero se vuelve a definir un vector de valores iniciales x_0, que por simplicidad y utilidad, será el mismo de la parte a.1)

$$x_0 = [a_0, \omega_0, \phi_0, b_0] = [a * 1.001, \omega * 1.001, \phi * 1.001, b * 1.001]$$

$$x_0 = [8.98374641, 1.01897537, 0.5370046, 0.32088405]$$

Nuestro $x_{objetivo}$ sigue siendo el mismo:

$$x = [a, \omega, \phi, b] = [8.97477164, 1.01795741, 0.53646814, 0.32056349]$$

Con nuestro vector de condiciones iniciales listo, volvemos a definir todas las funciones necesarias para implementar el metodo de la ecuación (4). Primero definimos un nuevo modelo $Mo(x, t)$, el cual es idéntico al utilizado en la parte a.1) ,y por ende, al de la ecuación (3). Una vez se tiene definido el modelo $M_o(x, t)$, nuevamente es directo definir $r(x)$ como $r_i(x) = y_i - M_o(x, t_i)$.

Como en esta parte no es necesario encontrar S(x), sólo nos queda encontrar el Jacobiano de r(x) que vuelve a ser el mismo Jacobiano de 750x4 de la parte a.1)

El Jacobiano de $r(x)$ es:

$$J(r(x)) = \begin{bmatrix} -\sin(\omega t[0] + \phi) & -a\cos(\omega t[0] + \phi)t[0] & -a\cos(\omega t[0] + \phi) & -t[0] \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ -\sin(\omega t[750] + \phi) & -a\cos(\omega t[750] + \phi)t[750] & -a\cos(\omega t[750] + \phi) & -t[750] \end{bmatrix}$$

Una vez tenemos todo lo necesario para implementar la ecuación (4), se procede a hacer casi la misma función recursiva de la parte a.1), con la diferencia de que la operación recursiva en este caso es la ecuación (4), y nuevamente haremos simples iteraciones del mismo para comparar sus funcionamientos.

Para implementar la función iterativa, llamada $newtonB(x_0, x_{objetivo})$ procedemos como sigue:

1) Calculamos $r(x_0)$, $J(x_0)$ y $S(x_0)$
2) Con estos valores realizamos la operación: $x_{k+1} = x_k - (J(x_k)^T J(x_k))^{-1} J(x_k)^T r(x_k)$
3) Si la diferencia (componente a componente) entre el vector iterado x_{k+1} y el vector objetivo x es pequeña, se retorna el valor de x_{k+1} que vendría a ser el x^* óptimo del ajuste e información acerca del número de iteraciones. En este paso puede imponer cualquier condición de parada que se estime conveniente, como que que el número de iteraciones sobrepase cierto límite (en este caso pondremos un máximo de 100 iteraciones), la condición de optimalidad de primer orden (ya que el problema es convexo e irrestricto) o que $\|\nabla f(x^*)\|$ sea muy pequeña.
4) Si no se cumple la condición impuesta en 3) el programa se ejecuta nuevamente desde 1) pero haciendo ahora $x_0 = x_{k+1} = x^*$ para que siga iterandose recursivamente hasta que cumpla con la condición del paso 3) y se detenga.

Si aplicamos esta iteración de $newtonB(x_0, x_{objetivo})$ para distintos valores de x_0 tenemos que:

Para $x_0 = x_{objetivo}$ el valor del ajuste tras 1 iteración es:
$x^* = [8.95474462, 1.01931466, 0.51372773, 0.32024964]$

Para $x_0 = 1.001 \cdot x_{objetivo}$ el valor del ajuste tras 1 iteración es:
$x^* = [8.95368004, 1.01931709, 0.51371617, 0.32024922]$

Para $x_0 = 1.1 \cdot x_{objetivo}$ el método sobrepasa el límite de 100 iteraciones

Para $x_0 = 7 \cdot x_{objetivo}$ el método sobrepasa el límite de 100 iteraciones

Para obtener más valores para comparar (y tener las mismas mediciones que en la parte a)), iteraremos mediante ciclos while 4 veces para $x_0 = 1.1 \cdot x_{objetivo}$, mientras que para $x_0 = 7 \cdot x_{objetivo}$ iteraremos 4 y 50 veces. Los valores de x^* para cada x_0 pasan a ser:

Para $x_0 = 1.1 \cdot x_{objetivo}$ el valor del ajuste tras 4 iteraciones es:
$x^* = [0.80345153, 6.27738031, -40.37310511, 0.29279909]$

Para $x_0 = 7 \cdot x_{objetivo}$ el valor del ajuste tras 4 iteraciones es:
$x^* = [-0.1212091, 7.22468375, 2.54801591, 0.29375685]$

Para $x_0 = 7 \cdot x_{objetivo}$ el valor del ajuste tras 50 iteraciones es:
$x^* = [0.12595837, 5.49549526, 10.89745566, 0.29381322]$

Si estos valores los comparamos con el valor original de x:

$$x = [a, \omega, \phi, b] = [8.97477164, 1.01795741, 0.53646814, 0.32056349]$$

y vemos el error porcentual de las aproximaciones del método de Newton para cada x_0, tenemos la siguiente tabla:

Tabla 4: Tabla de los errores porcentuales de x^* para distintos valores de x_0 utilizando el método de Gauss-Newton

Valor de x_0 utilizado	Error % para a	Error % para ω	Error % para ϕ	Error % para b	Iteraciones
$x_0 = x_{objetivo}$	0.2231	0.1333	4.2389	0.0979	1
$x_0 = 1.001 \cdot x_{objetivo}$	0.2350	0.1335	4.2410	0.0980	1
$x_0 = 1.1 \cdot x_{objetivo}$	91.0476	516.6643	7625.7227	8.6611	4
$x_0 = 7 \cdot x_{objetivo}$	101.3505	609.7235	374.9612	8.3623	4

Vemos que para utilizar este método, x_0 debe estar aún más cerca de $x_{objetivo}$ para encontrar el óptimo, ya que apenas se comienza a alejar un factor de $1.1 \cdot x_{objetivo}$ el error se dispara, a diferencia del método de Newton-Raphson de la parte a.1) y leastsq de la parte a.2) que seguían entregando valores óptimos cercanos al esperado. Nuevamente vemos que cuando se utiliza $x_0 = 7 \cdot x_{objetivo}$ el algoritmo no funciona. Tiene sentido que el método recursivo propuesto para esta parte tuviera más de 100 iteraciones para $x_0 = 1.1 \cdot x_{objetivo}$ y $x_0 = 7 \cdot x_{objetivo}$ ya que los errores son muy altos y difícilmente encontraría un óptimo.

Ahora haremos el proceso de iterar 1, 5 y 15 veces para $x_0 = 1.001 \cdot x_{objetivo}$, para ver cómo se comporta el método de Gauss-Newton a distintas iteraciones para el mismo x_0. Los resultados obtenidos fueron los siguientes:

Para $x_0 = 1.001 \cdot x_{objetivo}$ el valor del ajuste tras 1 iteración es:
$x^* = [8.95368004, 1.01931709, 0.51371617, 0.32024922]$

Para $x_0 = 1.001 \cdot x_{objetivo}$ el valor del ajuste tras 5 iteraciones es:
$x^* = [8.95537285, 1.01933231, 0.51344847, 0.32024498]$

Para $x_0 = 1.001 \cdot x_{objetivo}$ el valor del ajuste tras 15 iteraciones es:
$x^* = [8.95537285, 1.01933231, 0.51344847, 0.32024498]$

Vemos que el óptimo obtenido con el ajuste, x^*, se mantiene constante a partir de las 5 iteraciones aproximadamente al igual que sucedió al implementar el ajuste de la parte a.1). Esto nuevamente se puede deber al largo del paso α_k o al cumplimiento de alguna condición como la de Wolfe o la de Goldstein. Por otra parte notamos que tanto usando $newton(x_0, x_{objetivo})$ de la parte a) como $newtonB(x_0, x_{objetivo})$, como haciendolo converger mediante un while, el algoritmo de Newton converge rápidamente (en torno a las 5 iteraciones). Si intentamos ver como evolucionó el error con más iteraciones tenemos la siguiente tabla:

Tabla 5: Tabla de los errores porcentuales de x^* para un ajuste al mismo valor de x_0 pero distinta cantidad de iteraciones

Valor de x_0 utilizado	Error % para a	Error % para ω	Error % para ϕ	Error % para b	Iteraciones
$x_0 = 1.001 \cdot x_{objetivo}$	0.2350	0.1335	4.2410	0.0980	1
$x_0 = 1.001 \cdot x_{objetivo}$	0.2161	0.1350	4.2909	0.0993	5
$x_0 = 1.001 \cdot x_{objetivo}$	0.2161	0.1350	4.2909	0.0993	15

Si graficamos el resultado obtenido para el ajuste, en donde mediante $newtonB(x_0, x_{objetivo})$ se obtuvo el vector de condiciones iniciales óptimo x^* para $x_0 = 1.001 \cdot x_{objetivo}$. Usando $M_o(x^*, tdata)$ se obtuvo lo siguiente:

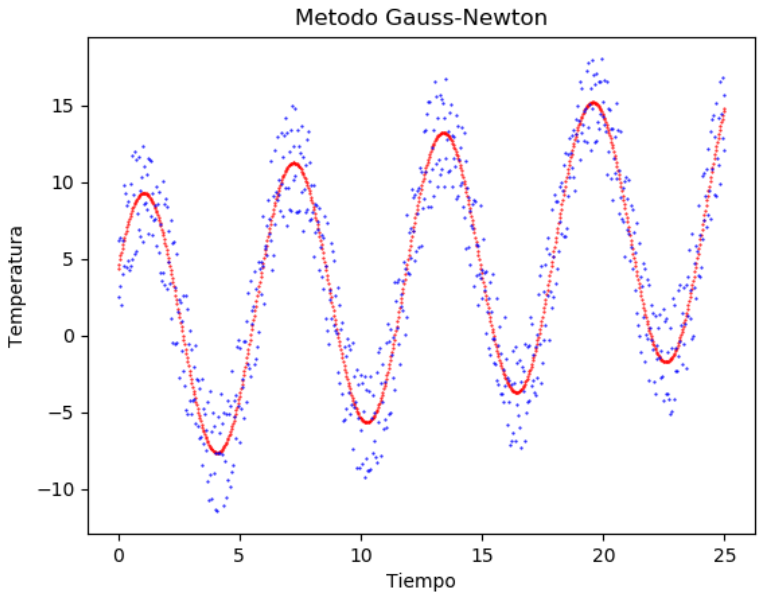

Figura 3: Ajuste mediante el método de Gauss-Newton para los datos, en azul ydata inicial y en rojo el ajuste.

4. Parte c)

Si comparamos los errores porcentuales obtenidos tanto por el método de Newton-Raphson en la parte a.1), como con leastsq en la parte a.2) y con Gauss-Newton en la parte b), para el mismo valor inicial $x_0 = 1.001 \cdot x_{objetivo}$, vemos que para valores muy cercanos al $x_{objetivo}$ los 3 presentan un nivel de error similar:

Método utilizado	Error % para a	Error % para omega	Error % para phi	Error % para b	Iteraciones
Newton-Raphson	0.2401	0.1339	4.2497	0.0991	1
leastsq	0.2161	0.1350	4.2909	0.0993	1
Gauss-Newton	0.2350	0.1335	4.2410	0.0980	1

La similitud de los 3 métodos se aprecia en las figuras 1, 2 y 3 donde los 3 métodos ajustan de manera similar los datos iniciales. Hay que tener en cuenta también que para valores de x_0 más alejados, la eficiencia de estos algoritmos no fue la mejor, siendo Gauss-Newton la peor en este sentido.

Un factor que podría estar afectando al error en los métodos es que invertir matrices numéricamente no es lo mejor, sobretodo cuando quedan matrices cercanas de ser singulares (no invertibles). Una forma para evitar este problema sería resolver el sistema de ecuaciones lineales asociado Ax=b.

Vimos que tanto el método de Newton-Raphson utilizado en la parte a.1), como leastsq en la parte a.2) y el método de Gauss-Newton utilizado en la parte b) sólo son útiles si se tiene un punto inicial cercano al óptimo, ya que el método sólo asegura convergencia local y para valores iniciales de x_0 muy lejanos al óptimo puede pasar cualquier cosa. Una alternativa son los algoritmos cuasi-Newton, que combinan de cierta manera los métodos de descenso y los métodos de Newton, y se acercan desde casi cualquier punto a una vecindad de una solución, después se podría utilizar Newton ya que, como hemos visto en esta tarea, converge bastante rápido (entre 1 y 5 iteraciones).